Linear Algebra in Cryptography: The Enigma Machine

David Moody, Haysun Choi, and Tyler Adams

It was a mathematical mystery to both the British and Polish intelligence services, before and during World War Two: the enigma machine. Developed by a German engineer, the Enigma machine was capable of transcribing text into a coded format that was indecipherable to everyone except the Germans. Therefore, Germany became very reliant on the enigma machine. It was used to encipher the communications that would only be deciphered by other militant Germans.

The enigma machine came into major use due to Germany's first cryptographic failures in World War One. Polish Intelligence had determined how to read some of the messages they intercepted. However, this became less feasible as the machine was made more complex through the addition of new circuits and plugs. These frequent modifications to the system made German communications less decipherable. And unlike during WWI, the British cryptanalysts were becoming less hopeful about their ability to break the enigma.

The Germans used this form of cryptography to communicate even during military engagements. Such was their confidence in the machine, which was not unfounded since the Germans took precautionary efforts to further complicate their communications. Essentially, they used the machine itself to create a day key that had to be double-checked by the receiver. The receiver could then adjust the scrambler settings on their end.

As the war broke out, cracking the code became a matter of international security. Its complete deciphering is said to have been a significant contribution in ending the war several years earlier and therefore essential in saving lives. The enigma machine's mathematical complexity can be explained to some degree using linear algebra concepts.

Introduction to Cryptography through a Linear Algebra Perspective

Linear algebra serves as a useful tool in cryptography, permitting the manipulation of multiple variables simultaneously to create a unique and reversible output. For example, one of the first known ciphers, the Caesar cipher, assigned each letter a number 0-25. Where 0 corresponds to A, B corresponds to 1, etc. An addition of 3 was performed (y+3=x), such that 0+3=3. That is, A corresponded to C, B to D, and so forth. The resulting jumble of letters represented a unique phrase and could be decoded using the the decryption key of x-3=y, where y was the original letter in the encrypted message. The matrix can represent an entire alphabet and its encrypted counterpart. Thus, linear algebra serves as a tool to manipulate simple shift ciphers, and more generally, affine ciphers, which multiply some integer k by y as well as perform a shift l (ky+l=x).

A Caesar shift is simple enough to envision without the aid of matrices as it simply pushes letters down the line. Even complete scramblings of the alphabet can be managed with a key relating the plain alphabet to the cipher alphabet, but delving into the effect that matrices can

have on alphabets will be useful in understanding more complicated ciphers. Let's consider a few different forms of matrices applicable to the subject. On the way, we will apply these matrices to the functions they represented on the Enigma machine.

Basic Matrices Used in Alphabet Manipulation

So, we begin with a basic Identity matrix, 26x26 so that we can map a letter to each position on the diagonal. This makes the Alphabet a vector of Dim 26. For the examples we'll use 6x6 because of space constraints, although the behavior translates to 26 dimensions.

A = XYZ =

1	0	0	0	0	0	a
0	1	0	0	0	0	b
0	0	1	0	0	0	c
0	0	0	1	0	0	d
0	0	0	0	1	0	e
0	0	0	0	0	1	f

The plugboard's matrix in the Enigma machine is simply the Identity matrix with swapped rows. So if we swap rows 1 and 3, now instead of row1 being 'a' it is 'c', and row3 is now 'a'.

Example matrix with swaps 1-3, 4-2, 5-6:

G1 =

0	0	1	0	0	0
0	0	0	1	0	0
1	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	1
0	0	0	0	1	0

Beyond simple swaps, a mechanical aspect of the machine went to work to increase deciphering difficulty. Every time a letter was entered, the first rotor shifts one position. After 26 clicks of the first rotor, the second rotor moves one position, and so on. There were three rotors in place at any given time. This movement (change of inputs essentially) can be represented by a shifting matrix, which in this case, is just the Identity shifted one.

Shft1 =

0	1	0	0	0	0
0	0	1	0	0	0
0	0	0	1	0	0
0	0	0	0	1	0
0	0	0	0	0	1
1	0	0	0	0	0

Which, when multiplied by G1 gives:

ans =

0	0	0	1	0	0
0	0	0	0	1	0
0	1	0	0	0	0
0	0	1	0	0	0
1	0	0	0	0	0
0	0	0	0	0	1

The shifting matrix, when seen alone, is equivalent to a Caesar cipher. Conveniently, all 25 shifting positions (26 if you count no shift at all) can be obtained by matrix multiplication of a single shifting matrix. That is, if our example matrix Shft1 were multiplied by itself, the resulting matrix would be a shifting matrix of *two* positions rather than 1, and so on.

When multiplied by Shft1, the rows of G1 shifted up 1 row, with the top row going to the bottom. Now when ‘a’ is pressed, it corresponds to ‘d’ in this case. Just as easily, ‘a’ could have been ‘e’ if the shift had been the other direction.

From here on let us refer to matrices that shift the alphabet as **shifting matrices**, those that swap letters as **swapping matrices**, and those with other rearranging functions simply as **scrambling matrices**. A key aspect of swapping matrices is that when one is multiplied to itself, the identity matrix is obtained. That is, for some swapping matrix W, W^2 = I. Therefore, W = W^-1. Now that we understand a little about how a matrix can be used to manipulate an alphabet, let’s take another look into the Enigma machine.

The Enigma as Matrices

The Enigma is not a monoalphabetic cipher, but a polyalphabetic cipher. This is a result of the rotor movements mentioned above. Imagine a linear function E maps R26 to R26, although technically it’s mapping vectors of letters rather than numbers. This function could be an

accurate model for the Enigma at a single keystroke, but another variable must be included in order to model the Enigma in all its glory: the character index. Let's call that c.

The machine works completely through mechanics and electrical circuits. When someone presses a key, it completes a circuit that travels through a plugboard that swaps several letters, then through one rotor (wheel in the diagram), through another rotor, through the last rotor, then across a reflector that sends the signal back through the rotors and plugboard. Finally, it reaches a lamp that indicates what letter the key has been encrypted to. Keep in mind that at any given keystroke, only a single key's signal is passing through the encryption, so the plugboard may or may not swap it with another key, etc.

Each of these stages can be represented with a matrix. The plugboard is simple because it doesn't move as a function of c (our character index). It could simply be represented by a static, swapping matrix P. Let a be the alphabet vector: a = {A, B, C, …, Z}. With only the plugboard in place, the Enigma function E would equal Pa. When the signal passes through the first rotor, it sends it to some random key. This rotor function could be represented by a scrambling matrix C. The signal passes through three of these. Let them be denoted as C1, C2, and C3. These rotors move according to the character index c (every time a key is pressed, one rotor moves one place, etc). However, the internal wiring of the rotor does not change, so the real effect that the movement has, is that it shifts the input by mechanically shifting the rotor. Let S be a shifting matrix shifted by one position. Now the function E should look something like C3 * S ^ (c / 26^2) * C2 * S ^ (c / 26) * C1 * S ^ (c) * P * a. Note that the / operator works without remainders because the shift must happen in terms of integers. That is, if c = 25, then c / 26 = 25 / 26 = 0, but if c = 26, then c / 26 = 26 / 26 = 1, etc. Also, the rotors may start at any setting, so we must include an offset f1, f2, f3 for the three rotors respectively: C3 * S ^ (c / 26^2 + f3) * C2 * S ^ (c / 26 + f2) * C1 * S ^ (c + f1) * P * a.

All that remains to define E is the reflector (matrix denoted by R), and a trip back through the rotors and plugboard. R, like P, is a swapping matrix. When the signal goes through the rotors a second time backwards, the wirings are swapped, so the matrices are all inverted. For convenience, let's denote every C * S ^ k pair as Q1, Q2, and Q3:

$$E = P * Q1^{-1} * Q2^{-1} * Q3^{-1} * R * Q3 * Q2 * Q1 * P * a$$

That's a lot of matrices, but from the diagram it looks like the whole procedure produces a swapping matrix: T goes to G, but G must go to T as well. Historically this made sense because in order to decrypt a message, the receiver needed only to type the encrypted message into the Enigma machine with the same settings, and would see the plaintext message. If the resulting matrix in E really is a swapping matrix, then it should be an inverse of itself, and it is! E * E = I.